Biopython Tutorial and Cookbook

Je Chang, Brad Chapman, Iddo Friedberg, Thomas Hamelryck, Michiel de Hoon, Peter Cock

Last Update{15 June 2008

Contents

Chapter 1

Introduction

1.1 What is Biopython?

The Biopython Project is an international association of developers of freely available Python (http://www.python.org) tools for computational molecular biology. The web site http://www.biopython.org provides

- Standalone Blast from NCBI
- Clustalw alignment program.

A standard sequence class that deals with sequences, ids on sequences, and sequence features.

Tools for performing common operations on sequences, such as translation, transcription and weight calculations.

6. *I looked in a directory for code, but I couldn't seem to nd the code that does something. Where's it hidden?*

   One thing to know is that we put code in `__init__.py`

Chapter 2

Quick Start { What can you do with Biopython?

2.4.2 Simple GenBank parsing example

Now let's load the GenBank le instead - notice that the code to do this is almost identical to the snippet

PubMed from NCBI { See Section 9.1 in the Cookbook for example code detailing how to use this.

SCOP

The code is these modules basically makes it easy to write python code that interact with the CGI scripts on these pages, so that you can get results in an easy to deal with format. In some cases, the results can be tightly integrated with the Biopython parsers to make it even easier to extract information.

2.6 What to do next

Now that you've made it this far, you hopefully have a good understanding of the basics of Biopython and are ready to start using it for doing useful work. The best thing to do now is to start snooping around in the source code and looking at the automatically generated documentation.

Once you get a picture of what you want to do, and what libraries in Biopython will do it, you should take a peak at the Cookbook, which may have example code to do something similar to what you want to do.

If you know what you want to do, but can't gure out how to do it, please feel free to post questions to the main biopython list (biopython@biopython.org). This will not only help us answer your question, it will also allow us to improve the documentation so it can help the next person do what you want to do.

Enjoy the code!

```
Seq('AGTACACTGGT', Alphabet())
>>> my_seq.alphabet
Alphabet()
```

```
>>>S>>S>t
>>>Alphab>>
>>> my_s>>Seq('AGTACACTGGT',
>>> my_s51
Seq('AGTACACTGGT',
>>> my_seq.alphabet
Suencesetastrtig51
```

>>>6>>>shou76-333(y27G9ocacts(y27G9eleme-)28(ts(y27G9of(y27G93cen)-76-n)-33ohab567G9in(y27G93cen)-76-namab

```
>>> my_seq = Seq("GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA", IUPAC.unambiguous_dna)
>>> standard_translator.translate(my_seq)
Seq('AIVMGR*KGAR', IUPACProtein())
>>> mito_translator.translate(my_seq)
Seq('AIVMGRWKGAR', IUPACProtein())
```

Notice that the default translation will just go ahead and proceed blindly through a stop codon. If you are aware that yt stostAotttfrau ttsts tsetup3o28(tunat)-

Chapter 4

The above example is repeated from the introduction in Section 2.4, and will load the orchid DNA sequences in the FASTA format file ls_orchid.fasta.

ID: Z78533.1
Name: Z78533
Desription: C.irapeanum 5.8S rRNA gene and ITS1 and ITS2 DNA.
/source=Cypripedium irapeanum
/taxonomy=['Eukaryota', 'Viridiplantae', 'Streptophyta', ..., 'Cypripedium']
/keywords=['5.8S ribosomal RNA', '5.8S rRNA gene', 'internal transcribed spacer', 'ITS1', 'ITS2']
/references=[...]

```
from Bio import Entrez
form Bio import SeqIO
handle = Entrez.efetch(db="protein", rettype="fasta", id="6273291")
seq_record = SeqIO.read(handle, "fasta")
handle.close()
print "%s with %i features" % (seq_record.id, len(seq_record.features))
```

Expected output:

```
gi|6273291|gb|AF191665.1|AF191665 with 0 features
```

Now let's fetch several records. This time the handle contains multiple records, so we must use the

You should recognise.ueecognisuognise.parsedgnise.ueegnisuASTd [(Agnise. legnisu)earliergnise.ingnise.Sogn1(ction)]TJ1

```
from Bio import SeqIO
from Bio.SeqUtils.CheckSum import seguid
seguid_dict = SeqIO.to_dict(SeqIO.parse(open("ls_orchid.gbk"), "genbank"),
                            lambda rec : seguid(rec.seq))
record = seguid_dict["MN/s0q9zDoCVEEc+k/IFwCNF2pY"]
print record.id
print record.description
```

That should have retrieved the record Z78532.1, the second entry in the le.

4.4 Writing Sequence Files

We've talked about using Bio.SeqIO.parse() for sequence input (reading les), and now we'll eeio.0ence inpb6(w)inpb
Bio.w(W)O.parse()

```
from Bio import SeqIO
```

Chapter 5

Sequence Alignment Input/Output

```
#=GS Q9T0Q8_BPIKE/1-52  AC Q9T0Q8.1
#=GS COATB_BPI22/32-83  AC P15416.1
#=GS COATB_BPM13/24-72  AC P69541.1
#=GS COATB_BPM13/24-72  DR PDB; 2cpb ; 1-49;
#=GS COATB_BPM13/24-72  DR PDB; 2cps ; 1-49;
#=GS COATB_BPZJ2/1-49   AC P03618.1
#=GS Q9T0Q9_BPFD/1-49   AC Q9T0Q9.1
#=GS Q9T0Q9_BPFD/1-49   DR PDB; 1nh4 A; 1-49;
#=GS COATB_BPIF1/22-73  AC P03619.2
#=GS COATB_BPIF1/22-73  DR PDB; 1ifk ; 1-50;
COATB_BPIKE/30-81             AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA
#=GR COATB_BPIKE/30-81  SS    -HHHHHHHHHHHHHH--HHHHHHHH--HHHHHHHHHHHHHHHHHHHHHH----
Q9T0Q8_BPIKE/1-52             AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA
COATB_BPI22/32-83             DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA
COATB_BPM13/24-72             AEGDDP...AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA
#=GR COATB_BPM13/24-72  SS    ---S-T...CHCHHHHCCCCTCCCTTCHHHHHHHHHHHHHHHHHHHHHCTT--
```

All that has changed in this code is the lename and the format string. You'll get the same output as

If you wanted to read this in using `Bio.AlignIO` you could use:

```
from Bio import AlignIO
alignments = AlignIO.parse(open("resampled.phy"), "phylip")
for alignment in alignments :
    print alignment
    print
```

This would give the following output, again abbreviated for display:

```
SingleLetterAlphabet() alignment with 5 rows and 6 columns
AAACCA Alpha
AAACCC Beta
SingleLetterAlphabet() alignment with 5 rows and 6 columns
AAACCA Alpha
ACCCCA Gamma
SingleLetterAlphabet() alignment with 5 rows and 6 columns
ACCAA[(AAA(Alpha)]TJ0-11.955Td[(AAACCC)-525(Beta)]TJ0-11.965Td[(ACCAA[(AAA(Gamma)]TJ0-11.955Td[(CCCCCA)
CCCA[(AAACAleta)]TJ0-11.955Td[(CCCACA)-525Epsilona
```

```
from Bio import AlignIO
alignments =AlignIO.parse(open("resampled.phy"), "phylip"")
aligs = alignment[-1]1
irst_(aligs)-525(=)-525(alignment[0]1)]T/626-11.9626Tf0-7.8225Td[(ts1)]T/F89.9626Tf0-8.3829Td[M(an)
alignmenhis clear. Hooovnertwhenaagenteals sequencealen(foma(n)458(hais)458(b)-)28(ena)458((usda)458((thren)4
```

structure. The most common such situation is when alignments have been saved in the FASTA file format. For example consider the following:

```
>Alpha
ACTACGACTAGCTCAG--G
>Beta
ACTACCGCTAGCTCAGAAG
>Gamma
ACTACGGCTAGCACAGAAG
>Alpha
ACTACGACTAGCTCAGG--
>Beta
ACTACCGCTAGCTCAGAAG
>Gamma
ACTACGGCTAGCACAGAAG
```

This could be a single alignment containing six sequences (with repeated identifiers). Or, judging from the identifiers, this is probably two different alignments each with three sequences, which happen to all have the same length.

What about this next example?

```
>Alpha
ACTACGACTAGCTCAG--G
>Beta
ACTACCGCTAGCTCAGAAG
>Alpha
ACTACGACTAGCTCAGG--
>Gamma
ACTACGGCTAGCACAGAAG
>Alpha
ACTACGACTAGCTCAGG--
>Delta
ACTACGGCTAGCACAGAAG
```

```
alphabet = Gapped(IUPAC.unambiguous_dna)

align1 = Alignment(alphabet)
align1.add_sequence("Alpha", "ACTGCTAGCTAG")
align1.add_sequence("Beta",  "ACT-CTAGCTAG")
align1.add_sequence("Gamma", "ACTGCTAGDTAG")

align2 = Alignment(alphabet)
align2.add_sequence("Delta",  "GTCAGC-AG")
align2.add_sequence("Epislon","GACAGCTAG")
align2.add_sequence("Zeta",   "GTCAGCTAG")

align3 = Alignment(alphabet)
align3.add_sequence("Eta",   "ACTAGTACAGCTG")
align3.add_sequence("Theta", "ACTAGTACAGCT-")
align3.add_sequence("Iota",  "-CTACTACAGGTG")

my_alignments = [align1, align2, align3]
```

Now we have a list of `Alignment` objects, we'll write them to a PHYLIP format file:

```
from Bio import AlignIO
handle = open("my_example.phy", "w")
SeqIO.write(my_alignments, handle, "phylip")
handle.close()
```

And if you open this file in your favourite text editor it should look like this:

```
 3 12
Alpha      ACTGCTAGCT AG
Td[28(12)]o0[28(12)] AG
Gamma      ACTGCTAGD] AG
 3 9
Delta      GTCAGC-AG
Epislon    GACAGCTAG
Zd[28(12)]GTCAGCTAG
 3 13
```

```
COATB_BPM1 AEGDDP---A KAAFNSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFTS
COATB_BPZJ AEGDDP---A KAAFDSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFAS
Q9T0Q9_BPF AEGDDP---A KAAFDSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFTS
COATB_BPIF FAADDATSQA KAAFDSLTAQ ATEMSGYAWA LVVLVVGATV GIKLFKKFVS

           KA
           RA
           KA
           KA
           KA
           KA
           RA
```

In general, because of the identi er limitation, working with PHYLIP le formats shouldn't be your rst choice. Using the PFAM/Stockholm format on the other hand allows you to record a lot of additional annotation too.

Chapter 6

BLAST

Hey, everybody loves BLAST right? I mean, geez, how can get it get any easier to do comparisons between one of your sequences and every other sequence in the known world? But, of course, this section isn't about

The code to deal with the WWW version of BLAST is found in the

```
>>> blast_results = result_handle.read()
```

Next, we save this string in a file:

```
>>> save_file = open("my_blast.xml", "w")
>>> save_file.write(blast_results)
>>> save_file.close()
```

The important point is that you do not have to use Biopython scripts to fetch the data in order to be able to parse it.

Doing things in one of these ways, you then need to get a handle to the results. In Python, a handle is just a nice general way of describing input to any info source so that the info can be retrieved using `read()` and `readline()`

```
>>> blast_records = list(blast_records)
```

Now you can access each BLAST record in the list with an index as usual. If your BLAST file is huge though, you may run into problems trying to save them all in a list.

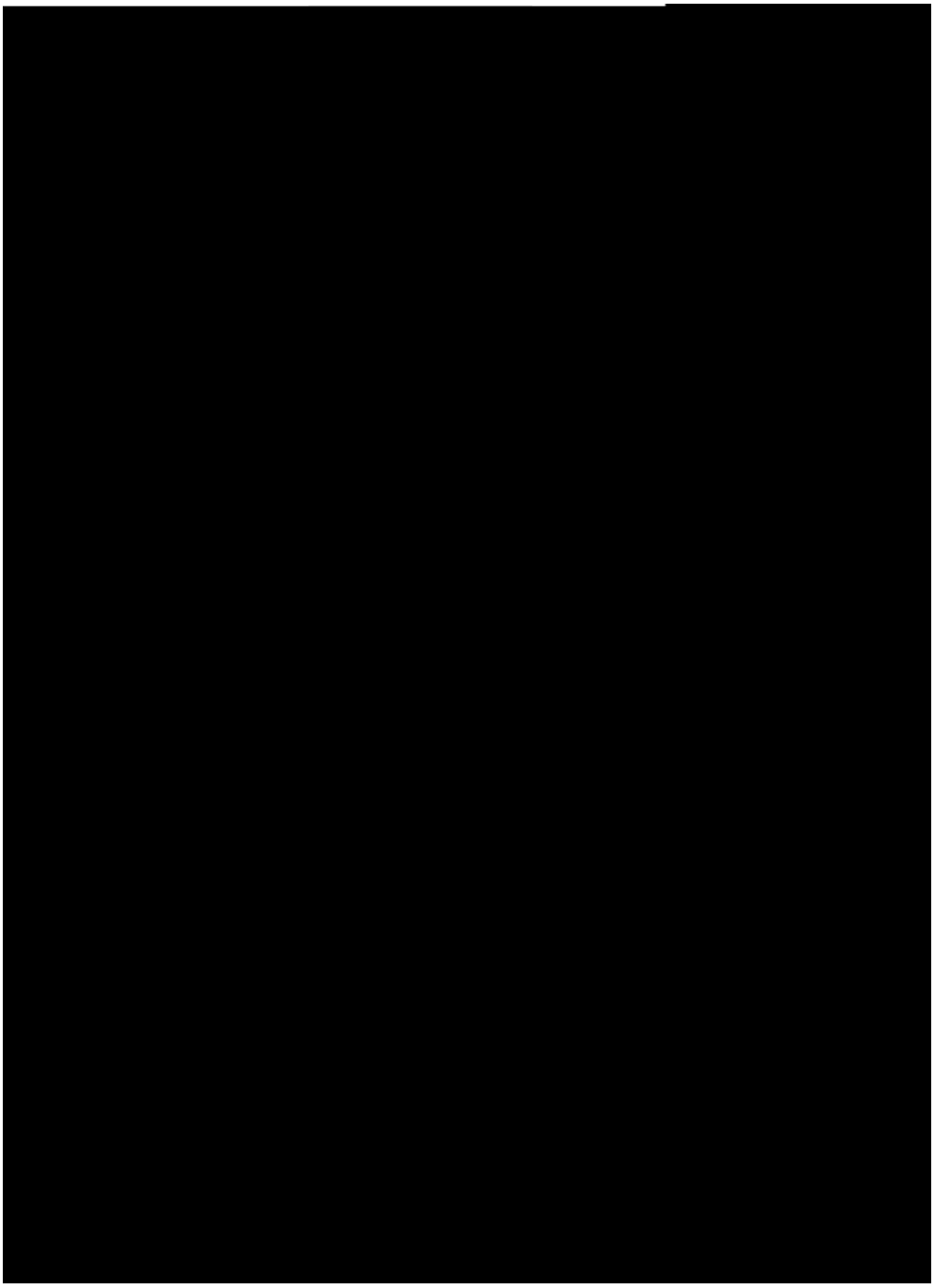

Figure 6.1: Class diagram for the Blast Record class representing all of the info in a BLAST report

The PSIBlast record object is similar, but has support for the rounds that are used in the iteration steps of PSIBlast. The class diagram for PSIBlast is shown in Figure 6.2.

6.6 Deprecated BLAST parsers

6.6.2 Parsing a file full of BLAST runs

6.7 Dealing with PSIBlast

Chapter 7

Make no more than one request every 3 seconds. This is automatically enforced by Biopython.

<DbName>mesh</DbName>
<DbName>ncbisearch</DbName>
<DbName>nlmcatalog</DbName>
<DbName>omia</DbName>
<DbName>omim</DbName>
<DbName>pmc</DbName>

7.3 ESearch: Searching the Entrez databases

To search any of these databases, we use `Bio.Entrez.esearch()`. For example, let's search in PubMed for publications related to Biopython:

```
>>> from Bio import Entrez
>>> handle = Entrez.esearch(db="pubmed", term="biopythonublicatiocil="A.N.O(thr@(exampl.com"0))]TJ0-11.
```

hinsoutpute, ye(PubMed)-834IDsn16377612e,whirchan

```
>>> from Bio import Entrez
>>> handle = Entrez.esummary(db="journals", id="30367", email="A.N.Other@example.com")
ove,-11.9-401.895Td[('30367')]TJ28(e)-334(can)-333(do)28(wnload)-333(GenBank)-334((record)333(57240072d)33
 Enton:>
>>> record = Entrez.read(handle)
>>> record[0]["Id"]
'30367'
>>> record[0]["Title-11.955Td[('30367')]dt"otatiorna>
    'I
ret>
```

Chapter 8

Swiss-Prot, Prosite, Prodoc, and ExPASy

```
>>> from Bio import SwissProt
>>> record = SwissProt.read(handle)
```

8.1.2 Parsing the Swiss-Prot keyword and category list

Swiss-Prot also distributes a file `keywlist.txt`, which lists the keywords and categories used in Swiss-Prot. The file contains entries in the following form:

```
ID   2Fe-2S.
AC   KW-0001
DE   Protein which contains at least one 2Fe-2S iron-sulfur cluster: 2 iron
DE   atoms complexed to 2 inorganic sulfides and 4 sulfur atoms of
DE   cysteines from the protein.
SY   Fe2S2; [2Fe-2S] cluster; [Fe2S2] cluster; Fe2/S2 (inorganic) cluster;
SY   Di-mu-sulfido-diiron; 2 iron, 2 sulfur cluster binding.
GO   GO:0051537; 2 iron, 2 sulfur cluster binding
HI   Ligand: Iron; Iron-sulfur; 2Fe-2S.
HI   Ligand: Metal-binding; 2Fe-2S.
CA   Ligand.
//
ID   3D-structure.
AC   KW-0002
DE   Protein, or part of a protein, whose three-dimensional structure has
DE   been resolved experimentally (for example by X-ray clvedpart.956Td[(CA)-1575(been)-525NMRuctureclu
//
```

8.3 Bio.Prosite.Prodoc: Parsing Prodoc records

In the Prosite example above, the `record.pdoc` accession numbers `'PDOC00001'`, `'PDOC00004'`, `'PDOC00005'` and so on refer to Prodoc records, which contain the Prosite Documentation. The Prodoc records are available from ExPASy as individual files, and as one file (`prosite.doc`) containing all Prodoc records.

```
...     handle = ExPASy.get_sprot_raw(accession)
...     record = SwissProt.read(handle)
...     records.append(record)
```

```
>>> from Bio import ExPASy
>>> from Bio import Prosite
>>> handle = ExPASy.get_prosite_raw('PS00001')
>>> record = Prosite.read(handle)
```

Finally, to retrieve a Prodoc record and parse it into a `Bio.Prosite.Prodoc.Record` object, use

```
>>> from Bio import ExPASy
>>> from Bio.Prosite import Prodoc
>>> handle = ExPASy.get_prosite_raw('PDOC00001')
>>> record = Prodoc.read(handle)
```

For non-existing accession numbers, `ExPASy.get_prosite_raw` returns a handle to an emptry string. When faced with an empty string, `Prosite.read` and `Prodoc.read` will raise a ValueError. You can catch

Chapter 9

Cookbook { Cool things to do with it

9.1 PubMed

The Bio.PubMed module uses Bio.Entrez internally to access the NCBI.

9.2.2 Parsing GenBank records

Or, using `Bio.SeqIO` instead (see Chapter

```
  G A T C
G 1 1 0 1
T 0 0 3 0
A 1 1 0 0
T 0 0 2 0
C 0 0 0 3
```

Let's assume we've got an alignment object called `c_align`. To get a PSSM with the consensus sequence along the side we first get a summary object and calculate the consensus sequence:

```
summary_align = AlignInfo.SummaryInfo(c_align)
consensus = summary_align.dumb_consensus()
```

Now, we want to make the PSSM, but ignore any `N`

9.3.5 Information Content

A potentially useful measure of evolutionary conservation is the information content of a sequence.

```
# get an alignment object from a Clustalw alignment output
c_align = Clustalw.parse_file("protein.aln", IUPAC.protein)
summary_align = AlignInfo.SummaryInfo(c_align)
```

Sections 9.3.1 and 9.3.2 contain more information on doing this.

Now that we've got our `summary_align` object, we want to use it to find out the number of times different residues substitute for each other. To make the example more readable, we'll focus on only amino acids with polar charged side chains. Luckily, this can be done easily when generating a replacement dictionary, by

```
>>> my_lom.print_mat()
D   6
E  -5   5
H -15 -13  10
K -31 -15 -13   6
R -13 -25 -14  -7   7
   D   E   H   K   R
```

Very nice. Now we've got our very own substitution matrix to play with!

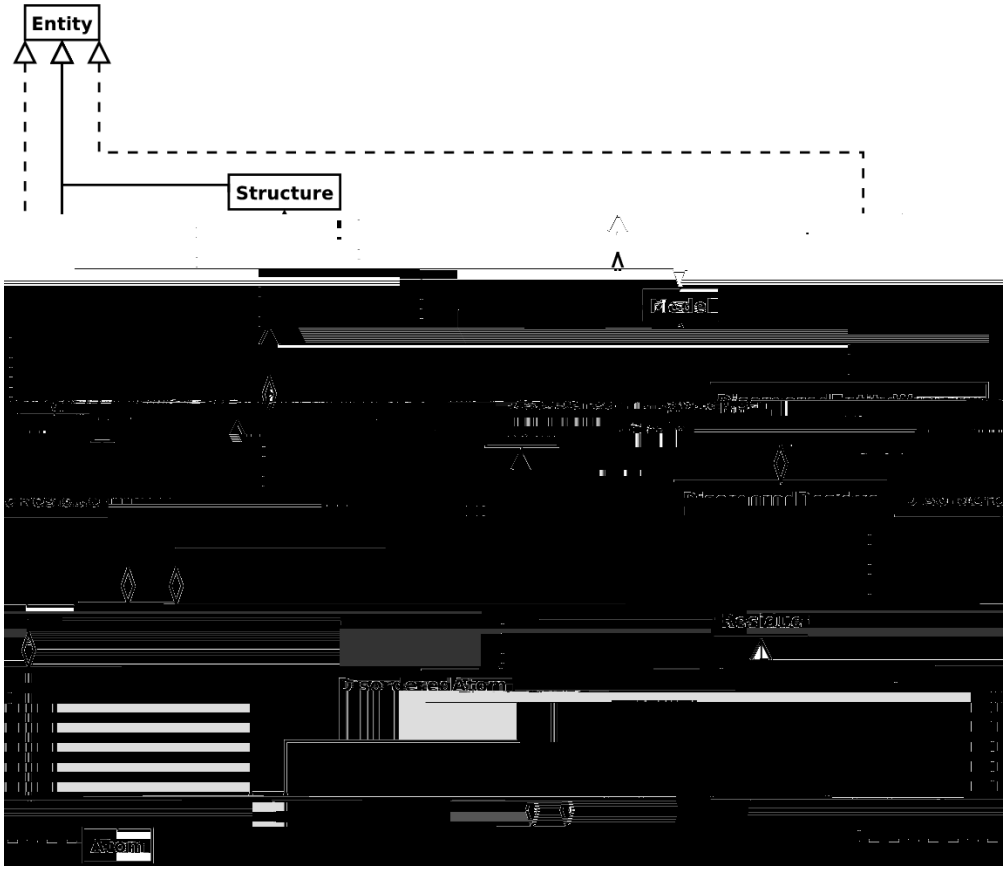

Figure 9.1: UML diagram of the SMCRA data structure used to represent a macromolecular structure.

Disordered atoms and residues are represented by DisorderedAtom and DisorderedResidue classes, which

9.7.1.1 Structure

The Structure object is at the top of the hierarchy. Its id is a user given string. The Structure contains

```
a.get_sigatm()     # std. dev. of atomic parameters
a.get_siguij()     # std. dev. of anisotropic B factor
a.get_anisou()     # anisotropic B factor
a.get_fullname()   # atom name (with spaces, e.g. ".CA.")
```

To represent the atom coordinates, siguij, anisotropic B factor and sigatm Numpy arrays are used.

9.7.2 Disorder

```
for residue in chain.get_list():
residue_id=residue.get_id()
hetfield=residue_id[0]
if hetfield[0]=="H":
print residue_id
```

rnameid=residue.ge[(rnameid())]TJ 0 -11.655 Td modelue_imodelin.get_id()
rname,inrseq51

if

ressu7

9.7.5.1.1 Duplicate residues One structure contains two amino acid residues in one chain with the same sequence identi er (resseq 3) and icode. Upon inspection it was found that this chain contains the residues Thr A3, . . . , Gly A202, Leu A3, Glu A204. Clearly, Leu A3 should be Leu A203. A couple of

9.7.6 Other features

There are also some tools to analyze a crystal structure. Tools exist to superimpose two coordinate sets

```
        ('Ind1', [(1, 2),    (3, 3), (200, 201)],
        ('Ind2', [(2, None), (3, 3), (None, None)],
    ],
    [
        ('Other1', [(1, 1),  (4, 3), (200, 200)],
    ]
]
```

9.8.2 Coalescent simulation

A coalescent simulation is a backward model of population genetics with relation to time. A simulation of

Figure 9.2: A bottleneck

```
from Bio.PopGen.SimCoal.Template import generate_simcoal_from_template

generate_simcoal_from_template('simple',
    [(1, [('SNP', [24, 0.0005, 0.0])])],
    [('sample_size', [30]),
    ('pop_size', [100])])
```

Executing this code snippet will generate a le on the current directory called simple

how to implement chromosome structures using the Biopython interface, not the underlying SIMCOAL2 capabilities.

We will start by implementing a single chromosome, with 24 SNPs with a recombination rate immediately on the right of each locus of 0.0005 and a minimum frequency of the minor allele of 0. This will be speci ed by the following list (to be passed as second parameter to the function generate_simcoal_from_template):

```
[(1, [('SNP', [24, 0.0005, 0.0])])]
```

This is actually the chromosome structure used in the above examples.

2. Compute average F_{st}. This is done by datacal inside FDist.

3. Simulate \neutral" markers based on the average F_{st} and expected number of total populations. This is the core operation, done by fdist inside FDist.

4. Calculate the con dence interval, based on the desired con dence boundaries (typically 95% or 99%). This is done by cplot and is mainly used to plot the interval.

5. Assess each marker status against the simulation \neutral" con dence interval. Done by pv. This is used to detect the outlier status of each marker against the simulation.

We will now discuss each step with illustrating example code (for this example to work FDist binaries have to be on the executable PATH).

The FDist data format is application speci c and is not used at all by other applications, as such you will probably have to convert your data for use with FDist. Biopython can help you do this. Here is an example converting from GenePop format to FDist format (along with imports that will be needed on examples further below):

sample_size Average number of individuals sampled on each population.

mut Mutation model: 0 - In nite alleles; 1 - Stepwise mutations

num_sims Number of simulations to perform. Typically a number around 40000 will be OK, but if you get a con dence interval that looks sharp (this can be detected when plotting the con dence interval computed below) the value can be increased (a suggestion would be steps of 10000 simulations).

The confusion in wording between number of samples and sample size stems from the original application.

A le named out.dat will be created with the simulated heterozygosities and F_{st}s, it will have as many lines as the number of simulations requested.

Note that fdist returns the average F_{st} that it was *capable* of simulating, for more details about this issue please read below the paragraph on approximating the desired average F_{st}.

The next (optional) step is to calculate the con dence interval:

```
cpl_interval = ctrl.run_cplot(ci=0.99)
```

```
sim_fst = ctrl.run_fdist_force_fst(npops = 15, nsamples = fd_rec.num_pops,
    fst = fst, sample_size = samp_size, mut = 0, num_sims = 40000,
    limit = 0.05)
```

The only new optional parameter, when comparing with run_fdist, is limit which is the desired maximum error. run

Chapter 10

Advanced

10.1 The SeqRecord and SeqFeature classes

Additionally, you can also pass the id, name and description to the initialization function, but if not they will be set as strings indicating they are unknown, and can be modi ed subsequently:

```
>>> simple_seq_r.id
'<unknown id>'
>>> simple_seq_r.id = 'AC12345'
>>> simple_seq_r.description = 'My little made up sequence I wish I could
write a paper about and submit to GenBank'
```

If you don't want to deal with fuzzy p sAons4(p)cuand4(p)cujus54(w)27(an)28(t)-454(to)-(t)-4umtt858(ou)-454(don'ju

Try to avoid anything which might be platform speci c, such as printing oating point numbers without using an explicit formatting string.

2. If the script requires les to do the testing, these should go in the directory [(Td01333s/Biospam[(2.)]TJ0 g 0 TJ 09.61

The `alphabet` optional argument is a string of all characters in the alphabet. If supplied, the order of letters along the axes is taken from the string, rather than by alphabetical order.

3. Usage

But you can supply your own `exp_freq_table`, if you wish

(d) Generating a substitution frequency matrix (SFM)

Use:

```
SFM = SubsMat._build_subs_mat(OFM,EFM)
```

Accepts an OFM, EFM. Provides the division product of the corresponding values.

(e) Generating a log-odds matrix (LOM)

Use:

```
LOM=SubsMat._build_log_odds_mat(SFM[,logbase=10,factor=10.0,round_digit=1])
```

i. -178(57-)]15.675 32.696.284f 73.225050SFM,EFM) or=10ed oo ofwn

And will be read using the `FreqTable.read_count(file_handle)`

Chapter 11

Where to go from here { contributing to Biopython

Macintosh { We would love to nd someone who wants to maintain a Macintosh distribution, and make

Chapter 12

Appendix: Useful stu about Python

If you haven't spent a lot of time programming in python, many questions and problems that come up in using Biopython are often related to python itself. This section tries to present some ideas and code that

```
>>> my_info = 'A string\n with multiple lines.'
>>> print my_info
A string
 with multiple lines.
>>> import cStringIO
>>> my_info_handle = cStringIO.StringIO(my_info)
>>> first_line = my_info_handle.readline()
>>> print first_line
A string

>>> second_line = my_info_handle.readline()
>>> print second_line
 with multiple lines.
```